Lightweight Semantic Annotation of Geospatial RESTful Services

Víctor Saquicela, Luis. M. Vilches-Blazquez, Oscar Corcho

Ontology Engineering Group, Departamento de Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{vsaquicela, lmvilches, ocorcho}@fi.upm.es

Abstract. RESTful services are increasingly gaining traction over WS-* ones. As with WS-* services, their semantic annotation can provide benefits in tasks related to their discovery, composition and mediation. In this paper we present an approach to automate the semantic annotation of RESTful services using a cross-domain ontology like DBpedia, domain ontologies like GeoNames, and additional external resources (suggestion and synonym services). We also present a preliminary evaluation in the geospatial domain that proves the feasibility of our approach in a domain where RESTful services are increasingly appearing and highlights that it is possible to carry out this semantic annotation with satisfactory results.

Keywords: REST, semantic annotation, geospatial RESTful services.

1 Introduction

In recent years, since the advent of Web 2.0 applications and given some of the limitations of "classical" Web services based on SOAP and WSDL, Representational State Transfer (REST) services have become an increasing phenomenon. Machine-oriented Web applications and APIs that are conformant to the REST architectural style [23], normally referred to as RESTful Web services, have started appearing mainly due to their relative simplicity and their natural suitability for the Web.

However, using RESTful services still requires much human intervention since the majority of their descriptions are given in the form of unstructured text in a Web page (HTML), which contains a list of the available operations, their URIs and parameters (also called attributes), expected output, error messages, and a set of examples of their execution. This hampers the automatic discovery, interpretation and invocation of these services, which may be required in the development of applications, without extensive user involvement.

Traditionally, semantic annotation approaches for services have focused on defining formalisms to describe them, and have been normally applied to WS-* service description formalisms and middleware. More recently, these (usually heavyweight) approaches have started to be adapted into a more lightweight manner for the semantic description of RESTful services [1, 5, 8]. However, most of the

processes related to the annotation of RESTful services (e.g., [2, 11]) still require a large amount of human intervention. First, humans have to understand the informal descriptions provided in the RESTful service description pages, and then the semantic annotation of RESTful services is done manually, with or without assistance.

In this paper, we address the challenge of automating the semantic annotation of RESTful services by: (1) obtaining and formalising their syntactic descriptions, which allows their registration and invocation, and (2) interpreting, and semantically enriching their parameters.

The main contribution of our work is the partial automation of the process of RESTful semantic annotation services using diverse types of resources: a cross-domain ontology, DBpedia (combined with GeoNames in the specific case of geospatial services), and diverse external services, such as suggestion and synonym services.

The remainder of this paper is structured as follows: Section 2 presents related work in the context of semantic annotation of WS-* and RESTful services. Section 3 introduces our approach for automating the annotation of RESTful services, including explanations on how we derive their syntactic description and semantic annotation. Section 4 presents the evaluation of our system in the context of these services from the geospatial domain. Finally, Section 5 presents some conclusions of this paper and identifies future lines of work.

2 Related work

Most research in the semantic annotation of RESTful services has focused on the definition of formal description languages for creating semantic annotations. The main proposed formalisms for describing these services are: the Web Application Description Language[1] (WADL), which describes syntactically RESTful services and the resources that they access; its semantic annotation extension [19]; MicroWSMO [3], which uses hREST (HTML for RESTful services) [3, 5]; and SA-REST [2, 8], which uses SAWSDL [1] and RDFa[2] to describe service properties.

From a broader point of view, the work done in the state of the art on Semantic Web Services (SWS) has mainly focused on WS-*services. OWL-S and WSMO are approaches that use ontologies to describe services.

Some authors propose the adaptation of heavyweight WS-* approaches to describe RESTful services. An example is proposed in [10], which makes use of OWL-S as the base ontology for services, whereas WADL is used for syntactically describing them. Then, the HTTP protocol is used for transferring messages, defining the action to be executed, and also defining the execution scope. Finally, URI identifiers are responsible for specifying the service interface.

Other approaches are more lightweight (e.g., [1, 2]). The authors advocate an integrated lightweight approach for describing semantically RESTful services. This approach is based on use of the hREST and MicroWSMO microformats to facilitate the annotation process. The SWEET tool [2] supports users in creating semantic

[1] http://www.w3.org/Submission/wadl/

[2] http://www.w3.org/TR/xhtml-rdfa-primer/

descriptions of RESTful services based on the aforementioned technologies. Unlike this work, our approach is focused on automating this process, and could be well integrated into this tool. Once the semantics of the RESTful service is obtained, this could be represented in any of the existing semantic description approaches, such as hREST, MicroWSMO, etc.

Finally, another approach for service description that focuses on automation, and hence can be considered closer to our work, is presented in [17]. This approach classifies service parameter datatypes using HTML treated Web form files as the Web service's parameters using Naïve Bayes.

3 An approach for the automatic semantic annotation of RESTful services

In this section, we present our approach, visualized in Figure 1, for automating the syntactic and semantic annotation of RESTful services. Our system consists of three main components, including invocation and registration, repository, and semantic annotation components, which are enriched by diverse external resources. Next, we briefly describe the different components, illustrating the descriptions with some sample services on the geospatial domain.

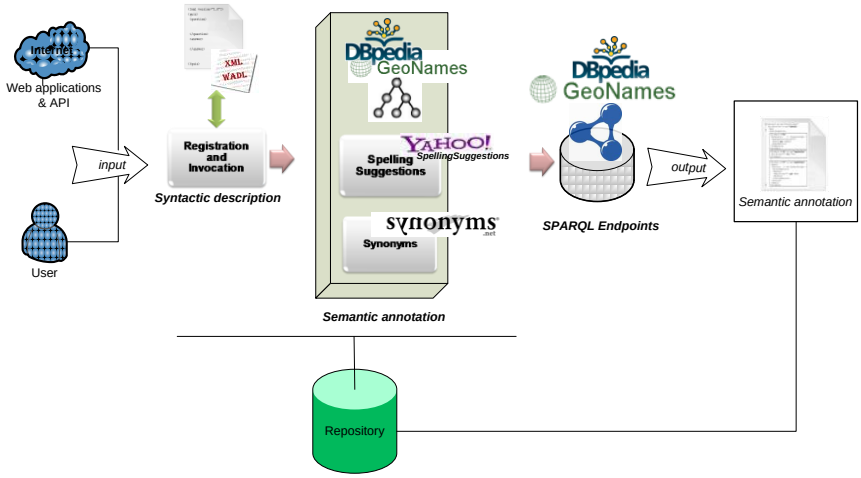

Figure 1. RESTful Service Semantic Annotation System

3.1 A sample set of RESTful services in the geospatial domain

Nowadays the largest online repository of information about Web 2.0 mashups and APIs is ProgrammableWeb.com. This aggregator site provides information on 5,401 mashups and 2,390 APIs that were registered between September 2005 and

November 2010. Mashups tagged as "mapping" represent a 44.5% mashups (2,403 mashups) of the listed ones, what represents the importance of geospatial information in the generation of these applications. With respect to APIs, GoogleMaps is the most used with an 89.4%, that is, this API is used on 2,136 mashups. These data show the importance of geospatial information in the context of the REST world. The following services, taken from the aforementioned site, are two representative geospatial RESTful services:

- **Service 1**. http://ws.geonames.org/countryInfo?country=ES

This service retrieves information related to a 'country'. More specifically, it returns information about the following parameters: 'capital', 'population', 'area' (km^2), and 'bounding box of mainland' (excluding offshore islands). In the specified URL, we retrieve information about Spain.

- **Service 2**. http://api.eventful.com/rest/venues/search?app_key=p4t8BFcLDtCzpxdS&location=Madrid

This service retrieves information about places (venues). More specifically, it returns parameters like: 'city', 'venue_name', 'region_name', 'country_name', 'latitude', 'longitude', etc. In the specified URL, we retrieve information about Madrid.

3.2 Syntactic description storing: Invocation and registration details into a repository

As aforementioned, RESTful services are normally described or registered in sites like *programmableWeb* by means of their URLs, plus some natural language descriptions, tags, and execution examples, if at all available. Hence, the first step in our system is to take as input the URL of an available RESTful service that is known by users (for instance, it has been discovered by a user by browsing this site, or it has been sent to the user by a friend).

In our system, the user adds the URLs of a service as a starting point, with the objective of obtaining automatically information related to it. Once the URLs is added, our system invokes the RESTful service with some sample parameters, obtained from the examples that are normally provided together with the URL (if this information is not available, our system cannot continue automatically without further human intervention), and analyzes the response to obtain a basic syntactic description of the parameter set, used as inputs and outputs.

In this process our system uses the Service Data Object[3] (SDO) API to perform the invocation of the RESTful service and determine whether it is available or not. SDO is a specification for a programming model that unifies data programming across data source types and provides robust support for common application patterns in a disconnected way [22]. The invocation process is performed as follows: first, it takes the input parameters and their values, which are given to the service as part of a URL.

[3] http://www.oasis-opencsa.org/sdo

Then, the system invokes the service that translates our "RESTful service call" into a query to a specific service, including the URL and related parameters.

The service invocation of a specific RESTful service may return diverse formats, such as JSON, XML, etc. In our work we use any of these formats, although for presentation purposes in this paper we will show how we handle XML responses. The results of a sample invocation of the services that we presented in section 3.1 are showed in Table 1.

Table 1. XML response of two sample RESTful services

Service 1	Service 2
<geonames> <country> <countryCode>ES</countryCode> **<countryName>Spain</countryName>** <isoNumeric>724</isoNumeric> <isoAlpha3>ESP</isoAlpha3> <fipsCode>SP</fipsCode> <continent>EU</continent> <capital>Madrid</capital> <areaInSqKm>504782.0</areaInSqKm> <population>40491000</population> <currencyCode>EUR</currencyCode> <languages>es-ES,ca,gl,eu</languages> <geonameId>2510769</geonameId> <bBoxWest>-18.169641494751</bBoxWest <bBoxNorth>43.791725</bBoxNorth> <bBoxEast>4.3153896</bBoxEast> <bBoxSouth>27.6388</bBoxSouth> </country> </geonames>	<venue id="V0-001-000154997-6"> <url>http://eventful.com/madrid/venues/la-ancha-/V0-001-000154997-6</url> **<country_name>Spain</country_name>** <name>La Ancha</name> <venue_name>La Ancha</venue_name> <description></description> <venue_type>Restaurant</venue_type> <address></address> <city_name>Madrid</city_name> <region_name></region_name> <region_abbr></region_abbr> <postal_code></postal_code> <country_abbr2>ES</country_abbr2> <country_abbr>ESP</country_abbr> <longitude>-3.68333</longitude> <latitude>40.4</latitude> <geocode_type>City Based GeoCodes</geocode_type> <owner>frankg</owner> <timezone></timezone> <created></created> <event_count>0</event_count> <trackback_count>0</trackback_count> <comment_count>0</comment_count> <link_count>0</link_count> <image></image> </venue> <venue id="V0-001-000154998-5">

These XML responses are processed using SDO, which enables to navigate through the XML and extract output parameters of each service[4]. The result of this invocation process is a syntactic definition of the RESTful service in XML, which can be expressed in description languages like WADL or stored into a relational model. Table 2 shows the different output parameters of each service, where we can observe by manual inspection that there is some similarity between diverse parameters (e.g., `countryName` and `country_name`) and that they return similar values (Spain). However, these parameters are written differently. These differences between parameters are described and dealt with in sections 3.3.1 and 3.3.2.

With URL and input/output parameters, we generate a WADL description that can be used as the input to the next process. Additionally, we register and store this description into a repository using an oriented-object model. This repository is implemented as a database that is specifically designed to store syntactic descriptions of RESTful services and parameters' values of invocations. We selected this storage model in order to increase efficiency in the recovery of the RESTful services.

[4] In the work reported here, we considered only XML tags with values.

Once the RESTful service is registered and the WADL description is generated, our system invokes the service without associated parameters. For example:

***Service 1'**. http://ws.geonames.org/countryInfo?
*This is an example of invocation (Service 1) without its associated parameters.

On the other hand, our system also considers service URLs as http://www.foo.org/weather/Madrid. These services belong to a specific RESTful entity and they are always invoked with its associated parameters.

In this way, the system invokes the service for retrieving a collection of instances (countries)[5] related to the service. The results of this invocation are stored into the oriented-object model. Thus, this process allows collecting additional information about a service (output parameters and instances), which is registered in our system, and retrieving it for future processes without the need to invoke the original service.

Table 2. Syntactic description of our two sample RESTful services

Service 1:
countryInfo($country,bBoxSouth,isoNumeric,continent,fipsCode,areaInSqKm,languages,isoAlpha3,countryCode,bBoxNorth,population,bBoxWest,currencyCode,bBoxEast,capital,geonameId,countryName)
Service 2:
rest/venues/search($location,$app_key,id,link_count,page_count,longitude,trackback_count,version,venue_type,owner,url,country_name,event_count,total_items,city_name,address,name,latitude,page_number,postal_code,country_abbr,first_item,page_items,last_item,page_size,country_abbr2,comment_count,geocode_type,search_time,venue_name)

3.3 Semantic annotation

Some of the difficulties that arise in the semantic annotation of RESTful services are briefly described in [1, 11]. In order to cope with them, we rely on techniques and processes that permit: a) semantic annotation using only the syntactic description of the services and their input/output parameters, or b) semantic annotation by identifying a set of example values that allow the automatic invocation of the service.

The starting point of the semantic annotation process is the list of syntactic parameters obtained previously (a WADL file or the model stored into a relational database). Once the RESTful service is syntactically described with all its identified input and output parameters, we proceed into its semantic annotation. We follow a heuristic approach that combines a number of external services and semantic resources to propose annotations for the parameters as shown in Figure 2. Next, we describe the main components of the semantic annotation.

[5] The results of this service invocation are available at
http://delicias.dia.fi.upm.es/RESTfulAnnotationWeb/RESTService1/RESTservice1.xml

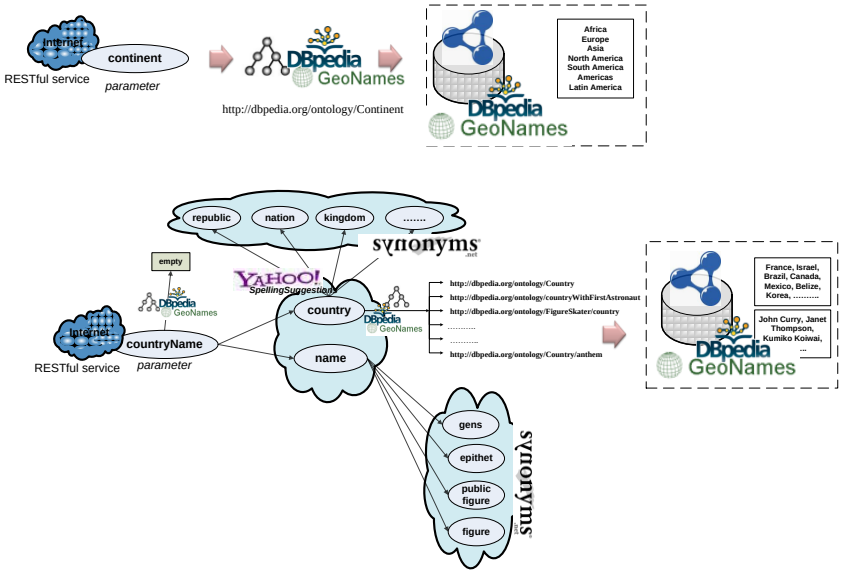

Figure 2. Semantic annotation process

3.3.1 A model for describing RESTful services

In order to describe semantically these services we define a model to represent the relationships of the different service parameters with the diverse resources used for semantic annotation. Some of the elements of this model are domain-independent, while others are domain dependent (in our examples we use these related to the geospatial domain, where we have performed our experiments in order to evaluate the feasibility of our approach). With respect to the domain-independent component, we use DBpedia, a community driven knowledge base, as the main source of background knowledge for supporting the semantic annotation process. This is complemented by the domain-dependent component. In the context of the shown examples, we use GeoNames[6] as a source related to geospatial information. This model (Figure 3) defines the following components:

- `Parameter`. This class provides a list of all parameters (inputs and outputs) collected from different services. Likewise, we search for additional information for each parameter, such as suggestions and synonyms, for enriching the initial description of parameters. The relation `hasCollection` relates `Parameter` with `DBpediaOntology`. Every parameter can be related to any number of DBpedia classes or properties (from 0 to N).
- `Ontologies`. This class contains classes and properties of the DBpedia and GeoNames ontology related to the parameters of each service. This class is related to the classes `DBpediaInstance` and `GeonamesInstance` by

[6] http://www.geonames.org/

means of the relation `hasCollection`. `Ontologies` can be related to any number of DBpedia or GeoNames instances (from 0 to N).

- `DBpediaInstance`. This class collects values from the DBpedia SPARQL Endpoint, where a parameter may have one o more associated resources.
- `GeonamesInstance`. This class collects geospatial information related to `latitude`, `longitude`, and `bounding box parameters` from a GeoNames SPARQL Endpoint.

The information related to each parameter of the RESTful service (semantic annotations) is stored only once in the system repository. By doing this, we avoid to duplicate information related to the same parameters, hence storing annotations independently of services and increasing the efficiency of our system.

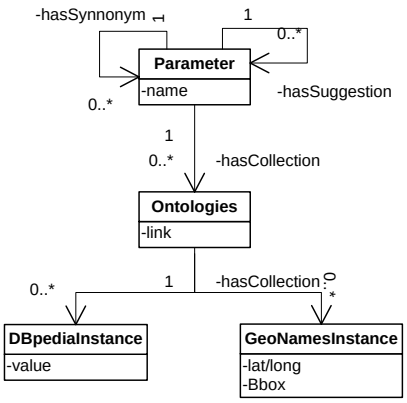

Figure 3. Model for the description of geospatial RESTful service parameters

3.3.2 Using semantic sources in the annotation process

At this stage, the list of syntactic parameters obtained previously is used to query the DBpedia and GeoNames SPARQL Endpoints (the latter is only used in the case of the geospatial domain) and retrieve associated results for each parameter, as follows:

- First, the system retrieves all the classes from the DBpedia ontology whose names have a match with each parameter of the RESTful service. In this matching process we test two different techniques:

  - On the one hand, our approach uses an exact match to compare parameters of RESTful service with the labels of the ontologies' classes and properties.
  - On the other hand, our approach uses a combination of various similarity metrics (Jaro, JaroWinkler and Levenshtein metrics)[7] to compare parameters with the labels of the elements of these ontologies. This proposal allows

[7] http://staffwww.dcs.shef.ac.uk/people/S.Chapman/stringmetrics.html

matching between strings such as `countryName`, `country_name`, or `country`, for example.

If the system obtains correspondences from the matching process, it uses these DBpedia concepts individually to retrieve samples (concept instances) from the DBpedia SPARQL Endpoint. Likewise, when a parameter matches an ontology class related to some geospatial information; such as latitude, longitude, or bounding box, our system retrieves samples from the GeoNames SPARQL Endpoint. The resulting information (RDF) is suggested automatically to the system and registered as a possible value for the corresponding parameter. When a parameter matches more than once in the DBpedia ontology, our system only considers those concepts that have information (instances), and automatically discards those ontology concepts without instances.

- Next, the system tries to find correspondences between parameters of the RESTful service and ontology properties. If the system obtains some correspondences, it uses these DBpedia properties individually to retrieve information of the DBpedia or GeoNames SPARQL Endpoint, as described above. Furthermore, this information is registered as a possible correct value for the corresponding parameter.
- Finally, with the obtained classes and properties, the system calls the DBpedia and GeoNames SPARQL Endpoints to retrieve values (instances) for those classes and properties, so that now we have possible values for them.

3.3.3 Enriching the semantic annotations

Our system looks for matches with DBpedia (and GeoNames) classes and properties. Hence it is possible to have parameters with not correspondences identified, since there are many lexical and syntactic variations that the parameter names may have, and because in some cases the information that is being requested may not be available in any of the external sources that are consulted. In order to annotate semantically the parameters that did not match any DBpedia resource, we use additional external services to enrich the results. Below we describe the main characteristics of the external services that we consider.

Spelling Suggestion Services

Web search engines (e.g. Google, Yahoo, and Microsoft) usually try to detect and solve users' writing mistakes. Spelling Suggestion services, also called "Did You Mean", are algorithms which aim at solving these spelling mistakes. For example, when a user writes 'countryName' these algorithms suggest 'country' and 'name' separately.

In our system we use the Yahoo Boss service[8] to retrieve suggestions about the parameters that we have obtained in the previous steps and for which we have not obtained any candidate in our semantic resources. Thus, for each parameter that the system did not find a correspondence with classes or properties in DBpedia (nor

[8] http://developer.yahoo.com/search/boss/boss_guide/Spelling_Suggest.html

GeoNames), this service is invoked to obtain a list of suggestions to query DBpedia (and GeoNames) again. The output is registered and stored into the repository. Following the previous example, the parameter 'countryName' is not found in the DBpedia ontology. Nevertheless, the added service allows separating this parameter in 'country' and 'name', and then it calls to the DBpedia SPARQL Endpoint with these new strings for obtaining results.

Synonym services

This external service[9] is incorporated into the system to retrieve possible synonyms for a certain parameter. This service tries to improve the semantic annotation process when our system does not offer results for the previous steps, that is, when we still have parameters in a RESTful service without any potential annotations.

As an example, we may have a parameter called 'address'. The invocation process uses the synonyms service to retrieve a set of synonyms of 'address' such as *extension*, *reference*, *mention*, *citation*, *denotation*, *destination*, *source*, *cite*, *acknowledgment*, and so on. These outputs are registered and stored into the repository, and then, the service calls to the DBpedia (and GeoNames) SPARQL Endpoints for results again.

Both spelling suggestion and synonym services use the matching process described in section 3.3.1 to find possible matches between the output of these services and the components of the used ontologies.

3.4 Checking the semantic annotation of RESTful services

In order to check the collected sample individuals and the initial semantic annotations obtained as a result of the previous process, our system invokes the RESTful services that were already registered in the repository (as we describe in Section 3.2) and validates the input and output parameters for checking which is the best option to describe each parameter.

For the validation of the **input parameters**, our system selects, for each parameter, a random subset of the example instances (of classes and/or properties) coming from the DBpedia (and GeoNames) ontology that we have obtained and registered before. Next, it makes several invocations of the RESTful service iterating over these registered values. The system does not check this with all the possible combination of collected instances for all parameters for two reasons: first, because of the combinatorial explosion that may be produced in such a case, and second because many RESTful services have invocation limitations.

When a service has one or more than one input parameter, the system obtains randomly some instances of this parameter for the validation process. Each parameter generates a collection (list) of instances from our repository. Then, the system joins instances to obtain a table of all combinations of each parameter. Likewise, the geospatial parameters, specifically latitude and longitude parameters, are combined to obtain some values (instances) that can be used for this invocation.

[9] http://www.synonyms.net/

If the service returns results from the invocation, then the service is considered as executable, and the corresponding annotations are marked as valid. If a service cannot be invoked successfully, the service is classified as non-executable and is automatically discarded from the list of services that can be automatically annotated.

For the validation of the **output parameters**, our system only takes into account executions with the correct inputs from the input sets that have been considered before. Next, the system compares the outputs obtained after execution with the information already stored in the repository due to the initial retrieval processes done before with DBPedia (and GeoNames), and external utility services. If the output can be matched, our system considers the output annotation as valid.

Finally, the correspondences that have been established between the different parameters of the RESTful service and the DBpedia (and GeoNames) ontology are registered and stored in the repository, so that they can be used later. In such a way, the RESTful service is annotated semantically and it will allow generating semantic descriptions or annotations of any of the types that were identified in the related work section (WADL, hREST, etc.). Table 3 provides an abbreviated form of this description for our exemplar service 1.

Table 3. Semantic annotation of a RESTful service

($country,http://www.w3.org/2003/01/geo/wgs84_pos#lat,http://www.w3.org/2003/01/geo/wgs84_pos#long,isoNumeric,http://dbpedia.org/ontology/Continent,fipsCode,http://dbpedia.org/property/areaMetroKm,languages,isoAlpha3,http://dbpedia.org/ontology/country,http://www.w3.org/2003/01/geo/wgs84_pos#lat,http://www.w3.org/2003/01/geo/wgs84_pos#long,http://dbpedia.org/ontology/populationDensity,http://www.w3.org/2003/01/geo/wgs84_pos#lat,http://www.w3.org/2003/01/geo/wgs84_pos#long,http://dbpedia.org/ontology/Currency,http://www.w3.org/2003/01/geo/wgs84_pos#lat,http://www.w3.org/2003/01/geo/wgs84_pos#long,http://dbpedia.org/ontology/capitalgeonameId,http://dbpedia.org/ontology/country)

4 Experimental results

In order to evaluate our approach in the geospatial domain we have used 60 different RESTful services found in *http://www.programmableweb.com/*, which we have selected randomly from those that were available and could be characterized to contain geospatial information by a manual lookup. The list of these services can be found in our experiment website[10]. In the syntactic registration of all these services in the system, by means of introducing the list of their URLs, our system successfully registered 56 of them into the repository (4 services could not be registered due to an

[10] http://delicias.dia.fi.upm.es/RESTfulAnnotationWeb/SourcesList/sources.ods

invocation error). As a result of this syntactic registration, the system has produced a complete list of 369 different parameters (52 input parameters and 342 output parameters), without duplications.

This analysis follows the three steps described in our semantic annotation process. First, our system identifies correctly 191 of 369 parameters by calling directly the DBpedia and GeoNames ontologies. Second, the system uses initial parameters plus the suggestion service and calls the DBpedia and GeoNames ontologies. In this case, it identifies 33 correspondences and adds 57 parameters to the initial ones. Third, the system uses the initial parameters plus the synonyms service, and calls the DBpedia and GeoNames ontologies. It identifies 126 correspondences and incorporates 1,147 additional parameters into the system. Finally, the system combines all the resources that result from the enrichment process and calls again the DBpedia and GeoNames SPARQL endpoint. Here it identifies 159 correspondences and adds 1,573 more parameters. A detailed view of these results is shown in Table 4.

Table 4. Enriching initial parameters with external resources

Attributes	**Total**	**Additional parameters**	**Matches** (*DBpedia and GeoNames ontologies*)
Initial parameters	369	-	191
Parameters + Suggestions	426	57	33
Parameters + Synonyms	1573	1147	126
Parameters + Suggestions + Synonyms	1573	1204	159

With respect to the validation of input parameters[11] (see Table 5), our system recognizes 152 inputs of the initial list, of which 76 parameters can be annotated automatically with the DBpedia (33 parameters) and GeoNames (45 parameters) ontologies.

Likewise, we have discovered with our evaluation that some other parameters are useless in terms of semantic annotation processes, since they refer to the navigation process through the RESTful service results or "special" parameters. These parameters (input/output) are not considered for this validation (nevertheless, they are considered to the invocation process), concretely 155 "special" parameters, for instance, `userID`, `api_key`, `page`, `total`, `hits`, etc.). These parameters were detected manually and a list of them is collected in this website[12]. Our system takes them out automatically from the service registration process[13].

One aspect of our system is that we cannot always guarantee a successful annotation, because in some cases the system cannot find any correspondence between the service parameters and the concepts or properties of the DBpedia or GeoNames ontologies. This is common, for instance, when parameter names are described by only one letter (e.g., `s`, `l` or `q`) and hence they are not sufficiently

[11] A detailed analysis on these input parameters is available at http://delicias.dia.fi.upm.es/RESTfulAnnotationWeb/inputs/inputs.ods

[12] http://delicias.dia.fi.upm.es/RESTfulAnnotationWeb/parameters/Parameters.ods

[13] This was not described in the process described in section 3 since we did not consider it relevant for the description of the whole process.

descriptive for our automated approach to find any correspondence. In our evaluation, we had 12 of this type of parameters. In these cases the parameters should be shown to users for a manual description of them.

In summary, for 56 of the 60 initial geospatial RESTful services we have obtained correct input parameter associations, except for 4 cases where we could not find any correspondence.

Table 5. Results of the input and output paremeters

RESTful Service	Total parameters	Annotated parameters	Annotated parameters (DBpedia)	Annotated parameters (GeoNames)	Special parameters	Service validation
Input parameters	152	76	33	45	73	56✓ 4×
Output parameters	862	315	202	113	299	-

With respect to the validation of output parameters[14] (see Table 5), our system recognizes 862 outputs that belong to the 56 services whose input parameters have been validated. This total of output parameters is divided into 315 whose correspondences can be found using DBpedia (202 parameters) and GeoNames (113 parameters) ontologies, and 391 (special (299) and not found (92) parameters) whose correspondences cannot be found.

Table 6. Output parameters metrics

RESTful Service	Found parameters	Not found parameters	Annotated	Not annotated	Right parameters	Precision	Recall
Output parameters	475	92	315	160	242	0.66	0.77

While in the context of the input parameters we are interested in determining whether we can call the service or not, in the case of output parameters, we are interested in the precision and recall metrics of the annotation process. Hence, we have generated a gold standard with the studied services in order to assign manually the annotations that have to be produced for all output parameters of these services, and we have performed an evaluation of the results obtained from the system for the parameters that are found. Regarding the parameters that are found, our system annotates 315 of them automatically, from which 242 parameters are annotated correctly according to the gold standard, while 160 parameters are not annotated. This provides us with an average value for precision equal to 0.66 and recall equal to 0.77 for both metrics.

To the best of our knowledge, there are no available results from existing research works to compare our results against. Likewise, these preliminary results prove the

[14] A detailed analysis on these output parameters is available at http://delicias.dia.fi.upm.es/RESTfulAnnotationWeb/ouputs/outputs.ods

feasibility of our system and highlight that its possible to carry out an assisted semantic annotation of RESTful services.

5 Conclusions and Future Work

In this paper we have proposed an approach to perform an assisted semantic annotation process of RESTful services. This process is implemented in a system that takes into account the DBpedia ontology and its SPARQL Endpoint, for general annotation, and GeoNames and its SPARQL Endpoint for geospatial specific results, as well as different external resources such as synonyms and suggestion services. We use combinations of these resources to discover meanings for each of the parameters of the RESTful services that a user may select and perform semantic annotations of them.

To illustrate our work and guide the explanations of the proposed semantic annotation process we have used two exemplary RESTful services related to the geospatial domain. Besides, we have presented some preliminary experimental results that prove the feasibility of our approach, at least in the geospatial domain, and show that it is possible to assist the semantic annotation of RESTful services, again at least in this domain.

Future work will focus on the development of a GUI that will ease the introduction of existing services by users for their semantic annotation, probably incorporated in any existing RESTful semantic annotation tool/utility suite. Furthermore, we also plan to make improvements to the proposed system through the analysis of instances retrieved in the matching process, so as to improve the results that have been demonstrated in our evaluation. In the same sense, we also aim at improving the SPARQL queries to DBpedia and other semantic resources associated or not to a specific domain, to better explore this resource in the annotation process, and optimize the use of suggestion and synonyms services. Finally, we will incorporate more specific domain ontologies in the semantic process for taking advantage of specific domain characteristics.

6 Acknowledgments

This work has been supported by the R&D project España Virtual (CENIT2008-1030), funded by Centro Nacional de Información Geográfica and CDTI under the R&D programme Ingenio 2010.

7 References

1. Maleshkova, M., Jacek Kopecky, and Pedrinaci, C. (2009) Adapting SAWSDL for Semantic Annotations of RESTful Services, Workshop: Beyond SAWSDL at OnTheMove Federated Conferences & Workshops, Vilamoura, Portugal

2. Maleshkova, M., Pedrinaci, C., and Domingue, J. (2009) Semantically Annotating RESTful Services with SWEET, Demo at 8th ISWC, Washington D.C., USA
3. Maleshkova, M., Gridinoc, L., Pedrinaci, C., and Domingue, J. (2009) Supporting the Semi-Automatic Acquisition of Semantic RESTful Service Descriptions, ESWC'09 Poster
4. Pedrinaci, C., Domingue, J., and Reto Krummenacher (2010) Services and the Web of Data: An Unexploited Symbiosis, Workshop: Linked AI: AAAI Spring Symposium "Linked Data Meets Artificial Intelligence"
5. Kopecký, J., Gomadam, K., and Vitvar T. (2008) hRESTS: An HTML Microformat for Describing RESTful Web Services. Web Intelligence 2008: 619-625
6. Lambert, D., and Domingue, J. (2008) Grounding semantic web services with rules, Workshop: Semantic Web Applications and Perspectives, Rome, Italy
7. Steinmetz, N., Lausen, H., Brunner, M. (2009) Web Service Search on Large Scale. ICSOC/ServiceWave 2009: 437-444
8. Lathem, J., Gomadam, K., and Sheth, A.P. (2007) SA-REST and (S)mashups : Adding Semantics to RESTful Services. ICSC 2007:469-476
9. García Rodríguez, M., Álvarez, J.M., Berrueta, D., and Polo, L. (2009) "Declarative Data Grounding Using a Mapping Language". Communications of SIWN, vol. 6, pp. 132-138.
10. Freitas Ferreira Filho, O., Grigas Varella Ferreira, M. A. (2009) Semantic Web Services: A RESTful Approach. IADIS Int. Conference WWW/INTERNET'09 Rome, Italy.
11. Alowisheq, A., Millard, D.E., and Tiropanis, T. (2009): EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. ISWC'09: 941-948
12. Alarcon, R., and Wilde, E. (2010) Linking Data from RESTful Services. LDOW10. Raleigh, North Carolina.
13. Lerman, K., Plangprasopchok, A., and Knoblock, C.A. (2007) Semantic Labeling of Online Information Sources. Int. J. Semantic Web Inf. Syst. 3(3): 36-56
14. Ambite, J.L., Darbha, S., Goel, A., Knoblock, C. A., Lerman, K., Parundekar, R., and Russ, T. A. (2009) Automatically Constructing Semantic Web Services from Online Sources. International Semantic Web Conference, pp. 17-32
15. Doan, A., Domingos, P., and Halevy, A. Y. (2001) Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. SIGMOD Conference 2001: 509-520
16. Doan, A., Domingos, P., and Halevy, A. Y. (2003) Learning to Match the Schemas of Data Sources: A Multistrategy Approach. Machine Learning 50(3): 279-301
17. Heß, A., and Kushmerick, N. (2003) Learning to Attach Semantic Metadata to Web Services, In Proc. Int. Semantic Web Conference.
18. Rahm, E., and Bernstein, P. (2001) "On matching schemas automatically," VLDB. Journal, vol. 10, no. 4.
19. Battle, R., and Benson, E. (2008) Brinding the semantic web and web 2.0 with Representational State Tranfer (REST). Web semantics 6, 61-69.
20. Braga, D., Ceri, S., Martinenghi, D., and Daniel. F. (2008) Mashing Up Search Services. IEEE Internet Computing 12(5):16-23.
21. Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y.H., Simmen, D., and Singh, A. (2007) Damia: a data mashup fabric for intranet applications. In Proceedings of the 33rd Int. Conference on VLDB'07 Endowment, pp. 1370-1373
22. Resende, L. Handling heterogeneous data sources in a SOA environment with service data objects (SDO) (2007) Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM (2007), 895-897.
23. Fielding, R. (2000) Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine.